intel®

APPLICATION NOTE

AP-311

October 1988

Using Macros in EPLD Designs

DANIEL E. SMITH
PROGRAMMABLE LOGIC APPLICATIONS
INTEL CORPORATION

Order Number: 292039-002

INTRODUCTION

The iPLS II (Intel Programmable Logic Software) Logic Optimizing Compiler includes a Macro Expander that supports the use of macros in EPLD designs. This application note shows how to use the TTL and EPLD Custom macros available from Intel with ADFs created by a text editor. Included are descriptions of macro file support, guidelines for using macros, and two design examples.

OVERVIEW

iPLS II allows designers to include macro calls in design files to implement common circuit functions. Macro calls are subsequently expanded by the LOC (Logic Optimizing Compiler) into ADF network and/or equation entries required to perform the desired functions. Use of macros allows designs to proceed at a high level, which simplifies and shortens the design process. Macros can be connected together or used in conjunction with standard iPLS II EPLD primitives. Designing with macros is analogous in many ways to using subroutines in software.

Macros can be used in ADFs (Advanced Design Files) created by a text editor, or by several schematic capture software products. This application note covers use of macros in ADFs created by a text editor. Macro support at this level includes the following:

- A TTL macro library (TTL.LIB) for designing with common TTL circuit equivalents
- An EPLD custom macro library (EPLDMAC.LIB) for designing with "generic" macros.
- A Macro Expander in the LOC that expands macro calls in ADFs with the contents of the corresponding macros from libraries.

Figure 1 shows text editor/ADF macro support for iPLS II. Note that the ADF can be created by any standard ASCII text editor (text editor supplied by user). Creation of user-defined macros is covered in application note, AP-312 "Creating Macros for EPLD Designs", order number 292040. Use of macros with schematic capture software is covered in the documentation for the respective software package.

This note discusses use of macros under the following headings:

- Macro Libraries, briefly describes the two libraries available from Intel.
- Using Macros, describes macro files, how to call macros, the process of macro expansion, calling multiple macro calls, and some basic guidelines to follow and pitfalls to avoid.
- Two examples showing use of TTL macros, and mixing macros and EPLD primitives.

MACRO LIBRARIES

Intel offers two macro libraries: a TTL Library and an EPLD Custom Library.

TTL Macro Library

A TTL macro library (TTL.LIB) is available from Intel to support design entry using familiar 74-series logic

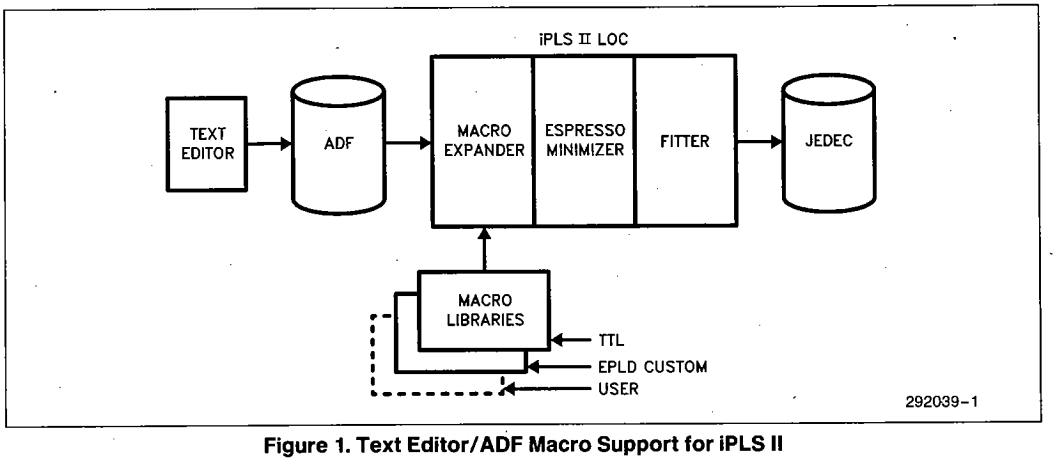


Figure 1. Text Editor/ADF Macro Support for iPLS II

devices. The library contains macros that implement the most widely used 74-series device functions as well as macros for some members of other logic families. Each device in the library is supported by a .DOC file. The .DOC file describes the macro syntax and lists any notable differences between the macro implementation and the TTL part.

EPLD Custom Macro Library

An EPLD custom macro library (EPLDMAC.LIB) is available from Intel to support design entry using groups of EPLD primitives or "generic" functions such as latches, registers, counters, decoders, etc.

USING MACROS

The iPLS II Macro Expander is automatically invoked by the LOC when an ADF is submitted to the compiler. When invoked, the Macro Expander identifies macro calls in ADFs, searches macro libraries for a corresponding macro, and expands the call with ADF network and equation entries from the macro file. The expanded file is then compiled normally.

Macro Files

Figure 2 shows the macro file for a 74138 TTL device, a commonly used one-of-eight decoder. Note that the first line contains the name and I/O signals for the device. Signals are listed in the order in which they appear on the actual TTL device, including VCC and GND (i.e., A = pin 1, B = pin 2, ..., VCC = pin 16). The sequence of signals in this line determines how the macro is "called" from an ADF.

Some of the macros in the TTL library have an "X" suffix appended to the filename, for example 74138X. This suffix indicates that the macro is device-specific (not supported on all EPLDs) or that there is some difference from the TTL device. This information is described in the .DOC file for each macro.

The second line of the macro file contains defaults for each input and place holders (blanks) for each output. The default for an input sets the input to an intelligent level (i.e., enables are enabled, clears, preset, loads are disabled, etc.).

Macro files can contain a Network section, an Equation section, or both. A Network section is not needed when the macro functions can all be implemented in Boolean equations. When used, the Network section contains EPLD design primitives. An Equations section is not needed when the macro functions can all be implemented in the Network section. Macro files end with the keyword "ENDEF".

Macro Calls

All macro calls appear in the Network section of an ADF. Macro calls use the same part/function name and signal sequence used on the first line of the macro file. The signal names in the macro and the macro call do not need to match, but the order of signals in the call is **crucial** to proper implementation of the macro function. For example, the macro call for the 74138 device could be any one of the following examples:

```
74138(A,B,C,G2A,G2B,G1,Y7,GND,Y6,Y5,
Y4,Y3,Y2,Y1,Y0,VCC)

74138(D1,D2,D3,EN1,EN2,EN3,07,GND,06,
05,04,03,02,01,00,VCC)
```

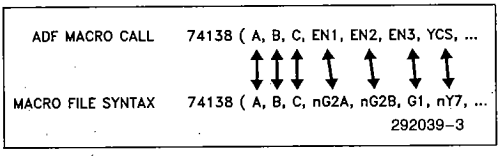
```
74138(A,B,C,nG2A,nG2B,G1,nY7,GND,nY6,nY5,nY4,nY3,nY2,nY1,nY0,VCC)
DEFAULT:(GND,GND,GND,GND,GND,VCC,,GND,,,,,,,VCC)

NETWORK:

EQUATIONS:

nY0   = !(!A * !B * !C * !nG2A * !nG2B * G1);
nY1   = !(A * !B * !C * !nG2A * !nG2B * G1);
nY2   = !(!A * B * !C * !nG2A * !nG2B * G1);
nY3   = !(A * B * !C * !nG2A * !nG2B * G1);
nY4   = !(!A * !B * C * !nG2A * !nG2B * G1);
nY5   = !(A * !B * C * !nG2A * !nG2B * G1);
nY6   = !(!A * B * C * !nG2A * !nG2B * G1);
nY7   = !(A * B * C * !nG2A * !nG2B * G1);

ENDEF
$
```

292039-2

Figure 2. Sample TTL Macro File (74138.DEV)

```
74138(A,B,C,ENA,ENB,ENC,Y7,GND,Y6,Y5,
Y4,Y3,Y2,Y1,Y0,VCC)
```

In each case, the part name corresponds to the macro part name. The names of the signals differ, but the order of signals match the macro. During processing, the Macro expander assigns node connections between the macro call and the macro file based on the positions of signals, not the names of the signals. For example, note the following macro call to macro file signal assignments:



TTL macro signals originating outside the target EPLD require a prior INPUT macro call in the Network section. All signals used as outputs require a prior OUTPUT macro call in the Network section. Figure 3 shows a sample ADF that uses the 74138 macro. Each input is listed in the INPUTS: declaration and has an INPUT macro call. Outputs are listed in the OUTPUTS: declaration and have OUTPUT macro calls. (EPLD INP and CONF primitive statements may also be used in place of INPUT and OUTPUT macro calls, if desired.)

Gate arrays support a much richer selection of input and output types than EPLDs. Gate array signals originating outside the target gate array device require the appropriate gate array input or output macro calls. When using gate array macros with EPLDs, the I/O macros are implemented in terms of EPLD primitives. Note that when designs targeted for gate arrays are partitioned for multiple EPLDs, many internal gate array signals are transformed into EPLD input and output signals. These signals must be supported by INPUT and OUTPUT macro calls.

Macro Expansion

The Macro Expander identifies and expands each macro call in an ADF with the corresponding macro definition from macro libraries (the TTL library in the case of the 74138). The Macro Expander searches libraries in the following order and in the directories listed:

- MACRO.LIB—first in the current directory, then in other directories specified by the DOS "PATH" variable.
- user libraries (filename.LIB)—names for user libraries are specified in the "IPLS" environment variable. If a pathname and filename are both specified (SET IPLS=C:\MACLIB\USR1.LIB;), the path is treated as an absolute path. If a filename alone is specified (set IPLS=USR1.LIB;), the Macro Expander searches for that library in the directories specified by the "PATH" variable. (IPLS can be set in an AUTOEXEC.BAT file.)
- TTL macro library (TTL.LIB)—first in the current directory, then in other directories specified by the DOS "PATH" variable.

```
YOUR NAME
YOUR COMPANY
DATE
1
A
5C060
One-of-Eight Decoder

OPTIONS: TURBO=OFF
PART: 5C060
INPUTS: A,B,C,G2A,G2B,G1
OUTPUTS: Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0

NETWORK:

INPUT(A,A)
INPUT(B,B)
INPUT(C,C)
INPUT(G2A,G2A)
INPUT(G2B,G2B)
INPUT(G1,G1)
OUTPUT(Y7,Y7)
OUTPUT(Y6,Y6)
OUTPUT(Y5,Y5)
OUTPUT(Y4,Y4)
OUTPUT(Y3,Y3)
OUTPUT(Y2,Y2)
OUTPUT(Y1,Y1)
OUTPUT(Y0,Y0)
74138(A,B,C,G2A,G2B,G1,Y7,GND,Y6,Y5,Y4,Y3,Y2,Y1,Y0,VCC)

END$
```

292039-4

Figure 3. ADF File Calling the 74138 Macro

- EPLD Custom macro library (EPLDMAC.LIB)—first in the current directory, then in other directories specified by the DOS "PATH" variable.
- reserved library (INTEL.LIB)—first in the current directory, then in other directories specified by the DOS "PATH" variable.

Only the first occurrence of a macro is used. The names TTL.LIB, EPLDMAC.LIB, and INTEL.LIB are reserved by Intel. They may not be used for user libraries and may not be specified in the "IPLS" variable. The "IPLS" variable can contain more than one library name. Each library can have an absolute path or can rely on the "PATH" variable to determine the search path.

The Macro Expander uses the ADF Network and Equation entries from the macro libraries and assigns the appropriate primitives for INPUT and OUTPUT calls. INP primitives are assigned to replace the INPUT macro calls. The OUTPUT calls are assigned primitives with output pins and output enables are supplied where needed.

Combination of primitives is automatically performed when needed. For example, when a feedback primitive such as a NORF feeds an output primitive such as a RONF, the Macro Expander combines the two primitives into a RORF. Combination of primitives conserves resources and results in the shortest possible delay path through the device.

During macro expansion, unused nodes are eliminated. For example, the VCC and GND nodes that correspond to TTL power and ground pins are eliminated. If an input node is not connected to a node in the ADF, the default value for that node is assigned from the

```
NETWORK:

%    INPUT(A,A)    %
%^% A=!INP(A)
%    INPUT(B,B)    %
%^% B=INP(B)
%    INPUT(C,C)    %
%^% C=INP(C)
%    INPUT(G2A,G2A)    %
%^% G2A=INP(G2A)
%    INPUT(G2B,G2B)        %
%^% G2B=INP(G2B)
%    INPUT(G1,G1)    %
%^% G1=INP(G1)
%    OUTPUT(Y7,Y7)    %
%^% Y7=CONF(Y7,VCC)
%    OUTPUT(Y6,Y6)    %
%^% Y6=CONF(Y6,VCC)
%    OUTPUT(Y5,Y5)    %
%^% Y5=CONF(Y5,VCC)
%    OUTPUT(Y4,Y4)    %
%^% Y4=CONF(Y4,VCC)
%    OUTPUT(Y3,Y3)    %
%^% Y3=CONF(Y3,VCC)
%    OUTPUT(Y2,Y2)    %
%^% Y2=CONF(Y2,VCC)
%    OUTPUT(Y1,Y1)    %
%^% Y1=CONF(Y1,VCC)
%    OUTPUT(Y0,Y0)    %
%^% Y0=CONF(Y0,VCC)
%    74138(A,B,C,G2A,G2B,G1,Y7,GND,Y6,Y5,Y4,Y3,Y2,Y1,Y0,VCC)    %

EQUATIONS:

%^% Y0=!(!A*!B*!C*!G2A*!G2B*G1);
%^% Y1=!(A*!B*!C*!G2A*!G2B*G1);
%^% Y2=!(!A*B*!C*!G2A*!G2B*G1);
%^% Y3=!(A*B*!C*!G2A*!G2B*G1);
%^% Y4=!(!A*!B*C*!G2A*!G2B*G1);
%^% Y5=!(A*!B*C*!G2A*!G2B*G1);
%^% Y6=!(!A*B*C*!G2A*!G2B*G1);
%^% Y7=!(A*B*C*!G2A*!G2B*G1);
```

292039-5

Figure 4. Network and Equations for 74138.SDF

DEFAULT: section of the macro file. Note, however, that the default value for each input in the macro file may be the value that disables the input or, for data inputs, is usually a logic 0. To be certain of the level used, specify a "VCC" or "GND" in the macro call for unused inputs.

The INPUT and OUTPUT calls and the original macro call are "commented out" by surrounding them with percent (%) signs. The %^% string is placed at the start of lines where primitives are created by the Macro Expander. The fully expanded file is written to the disk using the original filename and a .SDF extension. Figure 4 shows the Network and Equation sections for the 74138 SDF.

One final note with regard to compiling ADFs that use macros. Warning messages are typically encountered while compiling files that use macros. The most common message is "***WARN-XLT-Node Missing Destination". This message is displayed as unused nodes from a macro are deleted. For example, if a macro using a NOCF primitive is combined with a CONF and the original feedback is not needed, the warning is displayed as the feedback is deleted.

Multiple Macro Calls

The Macro Expander allows use of more than one macro in ADFs. Each macro must have its own call, even when the same macro is used more than once.

For example, to implement two 74138s, each case or "instance" must have its own call:

```
74138(A,B,C,G2A,G2B,G1,Y7,GND,Y6,Y5,
Y4,Y3,Y2,Y1,Y0,VCC)

74138(A,B,C,G3A,G3B,G1,YF,GND,YE,YD,
YC,YB,YA,Y9,Y8,VCC)
```

In this example, many of the inputs are routed to both devices. The Macro Expander automatically generates internal nodes for each instance of the macro. Each node is assigned a unique number based on the position of the macro in the Network section (i.e., . . 0140, . . 0141, etc. for nodes connecting to the 14th primitive in the Network section).

For traceability, you can define your own instance names for nodes of different macros by including the instance name in a comment immediately following the macro call. For example, to call two 74161 macros, one as Shift Register A and the other as Shift Register B, enter the calls as follows:

```
74161(CLR,CK,A,B,C,D,ENP,,LD,ENT,QD,
QC,QB,QA,RD1,) % SFA %

74161(CLR,CK,E,F,G,H,ENP,,LD,ENT,QH,
QG,QF,QE,RC2,) % SFB %
```

The Macro Expander uses the first three ASCII characters after the first percent sign (%), except for white space, to create instance numbers. For example, internal nodes for the first three signals of each macro call will be:

```
..SFAN1, ..SFAN2, ..SFAN3,

..SFBN1, ..SFBN2, ..SFBN3
```

where SFA/SFB are the user-defined instance names and N1, N2, N3 are the node numbers associated with each instance. For cases where no internal nodes numbers are generated, the Macro Expander simply ignores the instance name.

Outputs from one macro call can be used as inputs for other calls, as follows:

```
74138(A,B,C,G2A,G2B,G1,Y7,GND,Y6,Y5,
Y4,Y3,Y2,Y1,Y0,VCC)

74138(A,B,C,Y7,G3B,G1,YF,GND,YE,YD,YC,
YB,YA,Y9,Y8,VCC)
```

Here the Y7 output from the first decoder feeds an enable input of the second decoder.

Different macros are connected in the same manner. For example, the following macro calls connect the outputs from a 74138 decoder to the inputs of 74175 latches:

```
74138(A,B,C,G2A,G2B,G1,Y7,GND,Y6,Y5,
Y4,Y3,Y2,Y1,Y0,VCC)

74175(CLR,0Q,n0Q,Y0,Y1,n1Q,1Q,GND,CLK,
2Q, n2Q,Y2,Y3,n3Q,3Q,VCC)

74175(CLR,4Q,n4Q,Y4,Y5,n5Q,5Q,GND,CLK,
6Q, n6Q,Y6,Y7,n7Q,7Q,VCC)
```

Each decoder output is routed to a 74175 input. The 74175 macro produces both true and complement latched outputs.

Guidelines/Pitfalls

The following paragraphs discuss some general guidelines for using macros:

- Because the Macro Expander supports only one level of hierarchy, there is a tendency for p-terms to multiply quickly when several macros are connected together. In many cases, the total number of p-terms exceeds the capacity of the target EPLD. One method of avoiding problems with excessive p-terms is to route the outputs from a macro function through EPLD macrocells and use the feedbacks from the macrocells as inputs to the subsequent macro functions. This partitioning of functions trades off device resources for a lower p-term count.

- Implementation of some TTL macros requires primitives that are not supported on all devices. The .DOC file for a device notes any device dependency. In many cases, a modification to the basic TTL functions results in device independence. For example, a NOCF, which is not supported on all EPLDs, can be changed to a COIF, which is supported on all devices.
- Some macros use primitives that specify an output pin (COIF, CONF, RORF, etc.). These primitives must be supported with a signal name in the OUTPUTS: declaration and by an OUTPUT call in the Network Section of the ADF. Failure to provide this support causes the following error message during compilation:

```
***ERR-XLT-undeclared output name
```

If you encounter this error, check the macro file for output primitives that require ADF support.

Macro Usage Summary

ADF macro calls must observe the following guidelines:

- Macros are called from the Network Section of an ADF.
- The name in the call must match the name in the macro file (e.g., 74138 = 74138).
- All input and output pins on the target device must have both: (1) a corresponding signal name in the INPUTS: or OUTPUTS: declaration, and (2) a corresponding INPUT or OUTPUT macro call in the Network section. It is recommended that the same node name be used on both sides of each INPUT and OUTPUT macro call. This is required when macros containing CONFs are used. (EPLD INP and CONF primitives may also be used).
- All INPUT and OUTPUT calls in the Network section must precede any other macro call.
- Node connections within an ADF are made based on the names of the nodes.
- Connections between the macro call and macro files are based on the position of signal names in the call. Therefore, the sequence of inputs and outputs in a macro call must match the sequence of inputs and outputs in the corresponding macro file.

EXAMPLE 1: TTL MACROS

This section provides an example design using TTL macros.

Circuit

The design is a two-stage decoder using a 74138 macro and two 74139 macros. Figure 5 shows the schematic for the circuit. Each 74139 macro represents one half of a TTL 74139 device. Note that two of the outputs from the 74138 are routed back to enable the two 74139 decoders.

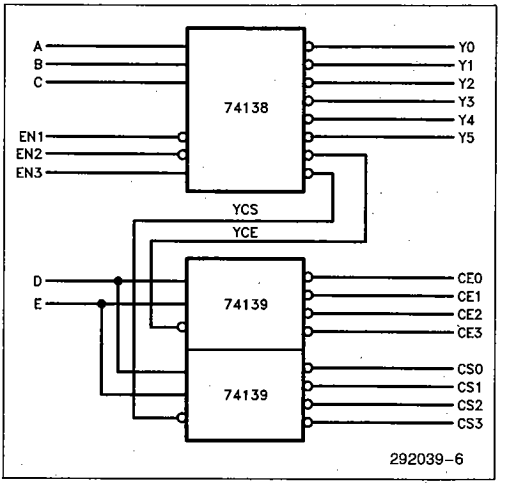


Figure 5. Schematic Diagram for Two-Stage Decoder

Figure 6 shows the ADF file containing the macro calls that implement the circuit. The two internal feedback signals (YCS and YCE) do not show up in the INPUTS: or OUTPUTS: declarations and are not represented by INPUT or OUTPUT calls in the Network section. The sequence of signals in the INPUTS: and OUTPUTS: declarations of the ADF is not important.

In the NETWORK: section, however, order is important. INPUT and OUTPUT calls must be listed before any other macro calls. This is a requirement of the Macro Expander. The sequence of signals within the ADF macro call is **critical**, as the Macro Expander automatically assigns macro call signals to macro file signals based on position.

Internal connections between macros are established by assigning the same name to the respective signals. For example, YCS in the 74138 macro call in Figure 7 represents the nY6 output from the 74138, while YCS in the 74139 macro call represents the 1G input to one 74139 decoder. Use of the same name establishes the connection. In the same manner, use of the signal name YCE connects the nY7 output from the 74138 to the 1G input of the second 74139.

```
DANIEL E. SMITH
INTEL CORPORATION
2/27/87
1
A
5C090
TWO-STAGE DECODER

OPTIONS: TURBO=OFF
PART: 5C090
INPUTS:  A,B,C,D,E,EN1,EN2,EN3
OUTPUTS: YO,Y1,Y2,Y3,Y4,Y5,CSO,CS1,CS2,CS3,CEO,CE1,CE2,CE3

NETWORK:

INPUT (A,A)
INPUT (B,B)
INPUT (C,C)
INPUT (D,D)
INPUT (E,E)
INPUT (EN1,EN1)
INPUT (EN2,EN2)
INPUT (EN3,EN3)
OUTPUT (YO,YO)
OUTPUT (Y1,Y1)
OUTPUT (Y2,Y2)
OUTPUT (Y3,Y3)
OUTPUT (Y4,Y4)
OUTPUT (Y5,Y5)
OUTPUT (CSO,CSO)
OUTPUT (CS1,CS1)
OUTPUT (CS2,CS2)
OUTPUT (CS3,CS3)
OUTPUT (CEO,CEO)
OUTPUT (CE1,CE1)
OUTPUT (CE2,CE2)
OUTPUT (CE3,CE3)
74138(A,B,C,EN1,EN2,EN3,YCS,GND,YCE,Y5,Y4,Y3,Y2,Y1,YO,VCC)
74139(YCS,D,E,CSO,CS1,CS2,CS3,GND,VCC)
74139(YCE,D,E,CEO,CE1,CE2,CE3,GND,VCC)

END$
```

292039-7

Figure 6. ADF File for Two-Stage Decoder Using TTL Macros

Sample Session

This session assumes familiarity with the iPLS II Logic Optimizing Compiler (LOC). For detailed information on the LOC, refer to Chapter 4 of the *iPLS II User's Guide,* order number: 450196. Proceed as follows to implement the TTL macro design shown here:

1. Use a standard ASCII text editor to create the ADF shown in Figure 7 under the name DECODE.ADF.
2. Invoke the iPLS II Menu by entering:

```
IPLS <Enter>
```

3. Invoke the LOC from the Main Menu by pressing <F4>.
4. Answer the LOC promts as follows:

```
Input Format?          <Enter>
File Name?             DECODE <Enter>
Minimization?          Y
Inversion Control?     N
LEF Analysis?          Y
Error Message File     <Enter>
```

The LOC then asks:

```
Do you wish to run under the above
conditions [Y/N]?
Enter: Y
```

The LOC expands the macros and compiles the expanded file to produce a JEDEC programming file (DECODE.JED), a utilization report file (DECODE. RPT), a minimized equation file (DECODE. LEF), and an error message file (DECODE.ERR). For tracability, a file called DECODE.SDF is created to show the expanded form of the ADF output by the Macro Expander.

5. The LOC terminates execution with the following message:

```
LOC cycle successfully completed
```

You can examine the LEF file to see the minimized form of the design. The LEF shows the EPLD primitives used to implement the design. Macro calls are not shown. If you wish, you can also use LPS (Logic Programmer Software) to program a part.

EXAMPLE 2: MIXING MACROS AND EPLD PRIMITIVES

This final example uses TTL macros together with standard EPLD primitives.

Circuit

The example circuit here is the 74138 macro used in example 1 with two of the outputs routed through additional combinatorial logic and RONF (Registered Output — No Feedback) primitives. Figure 7 shows the circuit. CS2 and CS3 are qualified by two additional inputs (RD* and WR*) to set or clear two latches. This is a configuration commonly used in microcomputer systems, where control signals are set and reset based on the address and command signals but not on a data value. A read to the port decoded by CS2 sets output LCS2 (Latched CS2) high. A write to that same port clears LCS2 low.

Figure 8 shows the ADF that implements the example circuit. This is the same ADF used in Figure 6, with the addition of several primitives and equations. The data inputs to both latches are tied to VCC. When RD* and the chip enable are both low, the respective clock signal goes low. As RD* or chip enable go high, the rising edge of the clock signal triggers the register, driving the output high.

Note that many Intel EPLDs do not support multiple product terms for register clocks. Therefore, the clock buffer primitive is driven by a macrocell configured as a COIF (Combinatorial Output—Input Feedback). Control signals (Clear and Preset) for many EPLDs also support only one product term. In this case, however, the NOR gate driving the clear input to the RONFs can be minimized to a single p-term. Thus a low on WR* and chip enable clears the respective latch to logic 0. (The intermediate macrocell for the Read function can be omitted for EPLDs that support two p-terms on register clocks.)

The connections between the TTL macros and the EPLD primitive are made by assigning the appropriate names to the input and output nodes. The CS2 and CS3 signals from the first example are no longer outputs, but are simply inputs to equations that feed the LCS2 and LCS3 RONF primitives.

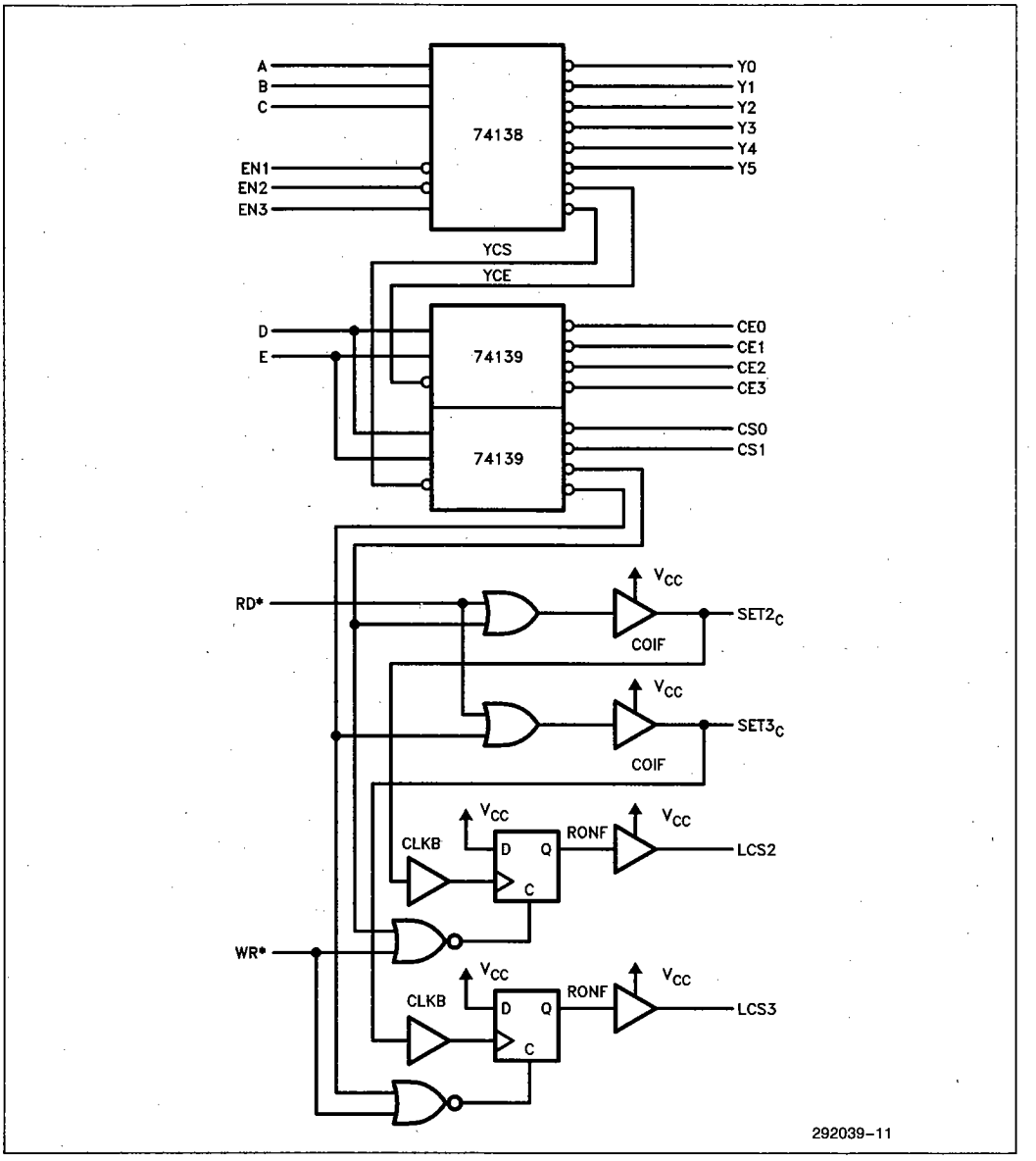


Figure 7. Schematic of Decoder Circuit with Latched Outputs

```
DANIEL E. SMITH
INTEL CORPORATION
2/27/87
1
A
5C090
DECODER WITH TWO LATCHED OUTPUTS

OPTIONS:  TURBO=OFF
PART:  5C090
INPUTS:  A,B,C,D,E,EN1,EN2,EN3,RD*,WR*
OUTPUTS:  SET2c,SET3c,Y0,Y1,Y2,Y3,Y4,Y5,CS0,CS1,LCS2,LCS3,CE0,CE1,CE2,CE3

NETWORK:

INPUT (A,A)
INPUT (B,B)
INPUT (C,C)
INPUT (D,D)
INPUT (E,E)
INPUT (EN1,EN1)
INPUT (EN2,EN2)
INPUT (EN3,EN3)
OUTPUT (Y0,Y0)
OUTPUT (Y1,Y1)
OUTPUT (Y2,Y2)
OUTPUT (Y3,Y3)
OUTPUT (Y4,Y4)
OUTPUT (Y5,Y5)
OUTPUT (CS0,CS0)
OUTPUT (CS1,CS1)
OUTPUT (CE0,CE0)
OUTPUT (CE1,CE1)
OUTPUT (CE2,CE2)
OUTPUT (CE3,CE3)
74138(A,B,C,EN1,EN2,EN3,YCS,GND,YCE,Y5,Y4,Y3,Y2,Y1,Y0,VCC)
74139(YCS,D,E,CS0,CS1,CS2,CS3,GND,VCC)
74139(YCE,D,E,CE0,CE1,CE2,CE3,GND,VCC)
RD = INP(RD*)
WR = INP(WR*)
LCS2 = RONF(VCC,SET2,CLR2,GND,VCC)
LCS3 = RONF(VCC,SET3,CLR3,GND,VCC)
SET2 = CLKB(SET2c)
SET3 = CLKB(SET3c)
SET2c,SET2c = COIF(ST2,VCC)
SET3c;SET3c = COIF(ST3,VCC)

EQUATIONS:

ST2 = RD + CS2;
CLR2 = /(WR + CS2);
ST3 = RD + CS3;
CLR3 = /(WR + CS3);

END$
```

292039-12

Figure 8. ADF File for Decoder with Latched Outputs

Sample Session

To implement this ADF in an actual session, follow the steps described for Example 1, substituting the name LDECODE for DECODE. iPLS II produces a JEDEC programming file (LDECODE.JED), a utilization report file (LDECODE.RPT), a minimized equation file (LDECODE.LEF), and an error message file (LDECODE.ERR). For traceability, a file called LDECODE.SDF is created to show the expanded form of the ADF output by the Macro Expander.